



xml from the inside out

Google®

[XML.COM](#) [WEB SERVICES.XML](#) [ORACLE NETWORK](#) [ORACLE.COM](#)
[Resources](#) | [Buyer's Guide](#) | [Newsletter](#) | [Safari Bookshelf](#)

Download Stylist Studio - The World's Best XML Editor

Creating Web Utilities Using XML::XPath

by [Kip Hampton](#)
 January 10, 2001

Mobile

Programming

Schemas

Style

Web

Web Services

Product Showcase

BEA Learning Channel

Tidy Up

The problem: You want to take advantage of the power and simplicity that XML tools can offer, but you face a site full of aging HTML documents. The solution: Convert your documents to XHTML and put Perl and XML::XPath to work.



A Code Library
 That Follows
 You Around

Try it *FREE*

Google™

HTML Tidy was created specifically to help HTML authors clean their markup. Now it offers the ability to translate HTML to XHTML. Use caution, however. Some of the errors that Tidy will fix may have been introduced by HTML authors in order to achieve a certain visual effect. Also keep in mind that the contents of most <script> elements will cause an XML parser to reject the parent

► **ESSENTIALS**
Annotated XML
What is XML?
What is XSLT?
What is XSL-FO?
What is XLink?
What is XML Schema?
What is XQuery?

document as not well-formed since some of the client-side script operators (>, <, and &) are special characters in the XML world. Unfortunately, most browsers do not support CDATA sections (the standard XML method for including text containing special characters) so the safest option is to move your client-side scripts out to separate files and include them using the script element's src attribute. Have a look at Simon St. Laurent's excellent XHTML notes for more suggestions. Be sure that your whole team has a chance to test the converted documents before putting them into production.

- [What are Topic Maps?](#)
- [What are Web Services?](#)
- [What are XForms?](#)
- [XSLT Recipe of the Day](#)

[Manage Your Account](#)
[Forgot Your Password?](#)

FIND
Search
Article Archive

Add the latest news
from XML.com to
your web site
[click here!](#)

▼ COLUMNS <taglines/>

Hacking the Library

Jon Udell

Practical XQuery

Rich Salz
Santé CVC

Standards Lowdown

XML-Deviants

▼ GUIDES

XML Resources
Buyer's Guide

Next we iterate over a hash of directories and files (where the directory is the key and the list of files in that directory is the value) adding elements to the output document along the way.

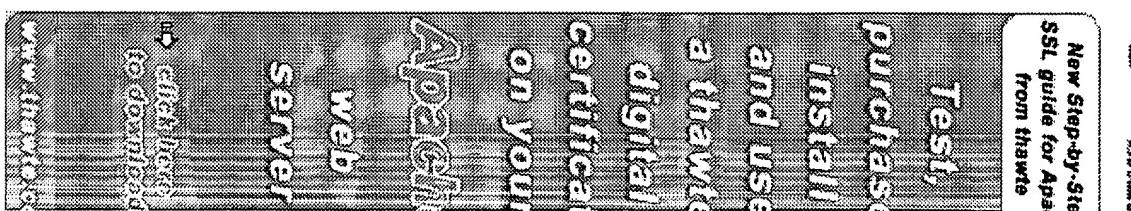
directory, parses all files with a given extension, and returns an XHTML sitemap to STDOUT. In addition to being a fairly useful tool, this script illustrates some of the DOM emulation functions available from within XML::XPath.

First we need to initialize new XML::XPath and XML::Parser objects and create two new XPath element nodes for our output document. Since the document we are creating is XHTML, we will name the root element "html" and its immediate child "body".

```
my $Sep = XML::Parser->new(Namespaces => 1);
my $xp = XML::XPath->new;
```

```
# create the root elements for the output tree
my $root = XML::XPath::Node::Element->new('html')
my $body = XML::XPath::Node::Element->new('body')
```

<http://www.xml.com/pub/a/2000/01/10/perlwebtools.html>



[Events Calendar](#)
[Standards List](#)
[Submissions List](#)

► TOOLBOX

Syntax Checker

```
# main loop
DIR: foreach my $directory ( sort (keys (%file_list)) ) {
    my $directory_container = XML::XPath::Node::Element->new('div');
    my $translated_directory = &translate_path($local_path,
                                              $directory);
    $translated_directory ||= '[root]';
    $directory_container->appendAttribute(
        XML::XPath::Node::Attribute->new('class', 'directory') );
    $directory_container->appendAttribute(
        XML::XPath::Node::Attribute->new('ID',
                                          $translated_directory) );
```

Developer Shed

- Open Source
 - ASP Help
 - Developer Tutorials
 - Computer Hardware
 - Search Engine Optimization
 - Scripts
- ```
my $directory_header = XML::XPath::Node::Element->new('h2');
$directory_header->appendChild(
 XML::XPath::Node::Text->new($translated_directory));
$directory_container->appendChild($directory_header);
my $file_list = XML::XPath::Node::Element->new('ul');
```

Note that the syntax `$parent->appendChild( XML::XPath::Node::Text->new('text') )` has the same effect as `$child = XML::XPath::Node::Text->new('text'); $parent->appendChild($child)` -- it's only a matter of code style.

Having created a container structure to represent the current directory, we loop through the list of files in that directory.

```
Traveling to a Tech Show?
New York City Hotels
Canada Hotels
Chicago Hotels
Hotel Discounts
Discount Hotels
California Hotels
Hotel Rooms
```

```
iterate over the files
FILE: foreach my $file (sort (@{$file_list->{$directory}})) {
 my ($root_node, $parser);
 my $list_item = XML::XPath::Node::Element->new('li');

 $xp->set_filename($file);
```

We want to include each document's title and description in the output, so we need to parse each file individually to extract that information. Rather than allowing the script to die from any errors encountered while parsing, we wrap the parse in an eval block and include any errors into the output for future reference.

```
eval {
```

XML.com  
supported by:

[Mortgages Online](#)  
[Payday Loans](#)

```

$parser = XML::XPath::XMLParser->new(
 filename => $xp->get_filename,
 parser => $sep);

$sroot_node = $parser->parse;
$xsp->find('/*', $sroot_node);

if ($@) {
 # if we get an error, include the details in the output.
 $list_item->appendChild(
 XML::XPath::Node::Text->new(
 "ERROR parsing file '$file': ". $@));
 $file_list->appendChild($list_item);
 next FILE;
}

```

If we get no errors during parsing, we translate the file path into a fully-qualified URL and extract the current file's title and description (if any). We then create a new hyperlink node, the <a> element. To this link node we add an href attribute node containing the URL and a text node containing the document's title.

```

my $translated_uri = &translate_path($local_path, $host_name, $file);

my $file_link = XML::XPath::Node::Element->new('a');
$file_link->appendAttribute(XML::XPath::Node::Attribute->new(
 'href' , $translated_uri));
my $title = $xp->find('/html/head/title',
 $root_node)->string_value || "Untitled Document
[$translated_uri]";

$file_link->appendChild(XML::XPath::Node::Text->new($title));

```

We append the <a> element to the list, <li>, item, as well as a text node containing the description. The last step is to add the item to the parent list. This cycle is repeated for each filename in the array.

```

my $desc = $xp->find(
 q{/html/head/meta[@content][@name="description"]},
 $root_node)->string_value || 'No Description';

add the child nodes to the anchor element
$list_item->appendChild($file_link);

```

```
$list_item->appendChild(XML::XPath::Node::Text->new(" - $desc"));
add the anchor to the output
$file_list->appendChild($list_item);
}

add the list to the directory container, and
the container to the main body,
$directory_container->appendChild($file_list);
$body->appendChild($directory_container);
}
```

As our script loops through each directory we add each "directory container" element (<div> and <ul>) elements to the <body> element.

```
add the "body" element to the root "html" node.
$sroot->appendChild($body);
print $sroot->toString;
```

Once we have processed all the directories, we append the <body> element to the root and print the result to STDOUT using the toString method.

A sample of the output might look like

```
<html>
<body>
<div class="directory" ID="/">
<h2>/</h2>

Welcome To My Site -
Top-level index page

</div>
<div class="directory" ID="/stuff">
<h2>/stuff</h2>

XML stuff - A
list of good XML links.
Llamas!! - Photos
from my recent llama packing trip.
Send Me A Message -
A simple contact form.

```

```
</div>
</body>
</html>
```

It may be tempting, given the simple format of this sitemap, to use a series of local variables and print statements to build the output. Experience shows, however, that this approach is like using regular expressions to parse XML: it might work for today's quick hack, but it will fail the instant your data become more complex. In general, I believe that Perl XMLers would do well to put aside the tendency to think of XML as text; instead, learn to see it as a textual representation of a tree-shaped data structure.

## Tool Two: Instant Site Search

If you've used the sitemap builder, you now have a clean index of your entire Web site that you can publish immediately. The site map can be useful for other ways too. Let's extend our sitemap just a bit and build a simple CGI search tool uses the generated sitemap as an index.

First, we need to expand the sitemap to include each document's meta keywords. To do this we add the following near the end of `sitemap.pl`'s FILE foreach loop, before we append the `<li>` to the file list.

```
my $keywords_text = $xp->find(
 q{!/html/head/meta[@name="keywords"]}),
 $root_node->string_value || '';
$keywords_node->appendChild(
 XML::XPath::Node::Text->new($keywords_text));
$list_item->appendChild($keywords_node);
```

With the list of keywords added to the sitemap we have only to write the CGI search script. Rather than having to search each document at runtime or dedicate database resources to allow visitors to search for what they're after, we use `XML::XPath` and a simple regular expression to search the title, description, and keywords from the sitemap.

```
use strict;
use CGI qw(:standard);
use XML::XPath;

initialize the objects and declare the path to the sitemap.
my $sitemap = "/usr/local/apache/htdocs/map3.html";
```

```
my $xp = XML::XPath->new(filename => $sitemap);
my $cgi = new CGI;

begin CGI output
print $cgi->header();

print $cgi->start_html(-title => 'Site Search');

print $cgi->h2('Search Our Site');

print $cgi->start_form(-method => 'POST',
 -action => 'mapsearch.cgi');

print $cgi->textfield(-name => 'search_string');

print $cgi->submit(-value => 'Search');

print $cgi->endform;

if the form has been submitted and the search box was filled in,
search the keywords, descriptions and titles for a match with
the input and print the link if a match is found.

if ($cgi->param('search_string')) {
 my $search_string = $cgi->param('search_string');
 my $match_count = 0;
 foreach my $page ($xp->findnodes('//li')->get_nodelist) {
 my $match_flag = 0;

 foreach my $field ('a', 'description', 'keywords') {
 $match_flag++ if
 $page->find($field)->string_value =~ /$search_string/gi;
 }

 if ($match_flag > 0) {
 print $cgi->a({href => $page->find('a/@href')}),
 $page->find('a')->string_value,
 $cgi->br,
 $match_count++;
 }
 }
}

print $cgi->b("No Matches Found for $search_string")
unless $match_count > 0;

print $cgi->end_html;
```

The output from this script might not win a Webby for design, but it clearly shows how to put the basic sitemap to more advanced uses. You could, for example, expand the range of the search by having sitemap.pl extract all of top-level HTML headers and appending the returned text nodes to an element called 'headers' in the output document. Tailoring the search to your specific needs is simply a matter of extracting the correct nodes from the site's documents into the sitemap.

## Summing Up

The tools covered in this article are simple examples designed to show how easy it is to use the combined power of XHTML, Perl, and XML::XPath to create Web utilities. Hopefully these samples have served to encourage you to experiment with your own Perl XML inventions, or, at the very least, have expanded your definition of XML Web development. Thanks go to [Kendall Clark](#) for bringing up the notion of using XPath expressions as the basis for a Web search and extraction tool.

## Resources

- [Download sample code](#)
- [HTML Tidy](#)
- [XHTML Resources at the W3C](#)
- [XHTML Notes From Simon St.Laurent](#)

Ads by Google			
<a href="#">Powerful XML Tools</a> Create, Edit, Transform XML assets Download Now - Free Trial <a href="http://www.snapbridge.com">www.snapbridge.com</a>	<a href="#">Free XPath Examples</a> XPath Made Easy with XML Spy Visualize/Analyze XPath, Free D/L <a href="http://www.altova.com">www.altova.com</a>	<a href="#">Output for XML</a> WYSIWYG, many output formats Open standards, multi-platform <a href="http://www.scriptura-xsl.com">www.scriptura-xsl.com</a>	<a href="#">VisiBone XHTML Cheatsheet</a> CSS cross-referenced with XHTML. Color-coded browser compatibility. <a href="http://visibone.com/html/">visibone.com/html/</a>

[Contact Us](#) | [Our Mission](#) | [Privacy Policy](#) | [Advertise With Us](#) | [Site Help](#) | [Submissions Guidelines](#)

Copyright © 2004 O'Reilly Media, Inc.